Protection from Unresponsive Flows with Geometric CHOKe

Addisu Eshete and Yuming Jiang
Centre for Quantifiable Quality of Service in Communication Systems*
Norwegian University of Science and Technology, Trondheim, Norway
addisu.eshete@q2s.ntnu.no ymjiang@ieee.org

***Abstract*—This paper proposes a simple and stateless active queue management (AQM) scheme, called geometric CHOKe (gCHOKe), to protect responsive flows from unresponsive ones. The proposed gCHOKe has its root on and is a generalization of the original CHOKe. It provides an extended power of flow protection, achieved by introducing an *extra flow matching* trial upon each successful matching of packets. Compared to the plain CHOKe, analysis and simulation show that gCHOKe can achieve over 20% improvement in the bounds of both bandwidth and buffer space used by an aggressive flow. In addition, up to 14% of the total link capacity can be saved from the unresponsive flow, allowing responsive or rate-adaptive flows to obtain a better share of resources in the router.**

***Index Terms*—AQM, RED, CHOKe, TCP, Flow Protection**

1. Introduction

A classical way to enforce fairness in the Internet has been with the help of congestion avoidance algorithms implemented at the end hosts. Such end-to-end (e2e) schemes, however, provide no incentives for users, and an unresponsive end host or flow, which (intentionally or unintentionally) lacks e2e congestion avoidance scheme, may end up with a lion share of bandwidth in the network. Therefore, they should be complemented with router mechanisms that can help enforce fairness by identifying unresponsive flows and restricting their share of bandwidth. Generally speaking, there are two types of router schemes for this purpose: (1) per flow scheduling schemes (e.g., SFQ [1]) and, (2) active queue management schemes (e.g., RED [2]). The former type schemes are usually stateful and require the router to identify flows and to dynamically manage possibly huge number of queues. These requirements impose complexity and scalability issues in implementation. An active queue management (AQM) scheme normally only has and operates on one queue, and is more scalable and easier to implement.

Among the existing AQM schemes, RED (Random Early Detection) [2] is perhaps the most widely known. RED is a stateless scheme. A congested RED router randomly drops incoming packets with a certain probability. Since packets are admitted to or dropped by the RED queue probabilistically, both the packet drop and admission histories form an unbiased statistics about the state of affairs regarding the rate of active flows. As a result, a flow is more likely to have packet drop or admission tallies *proportional* to its rate of arrival. This implies that *even with RED, an unresponsive flow can still get high bandwidth share and starve responsive flows*. To deal with this problem, several RED variations have been proposed, such as FRED (Flow Random Early Drop) [3] and RED-PD (RED with Preferential Dropping) [4]. These schemes leverage the recent flow history to *identify and control* high bandwidth flows. Particularly, these schemes keep state for the high-bandwidth (and possibly unresponsive) flows with which additional controlling of their throughput is enforced. However, while appealing, these schemes become stateful or need to maintain and make use of *partial flow state* [4].

In [5], an original and highly novel scheme, called CHOKe, is proposed to protect responsive flows from unresponsive ones. It can easily be implemented by a simple tweaking of the RED algorithm. CHOKe is completely stateless with no need of storing or maintaining flow level state at the router. It uses the recent admissions, i.e. packets currently queued in the buffer, to penalize the high bandwidth flows. Specifically, *"when a packet arrives at a congested router, CHOKe draws a packet at random from the FIFO buffer and compares it with the arriving packet. If they both belong to the same flow, then they are both dropped; else the randomly chosen packet is left intact and the arriving packet is admitted into the buffer with a probability that depends on the level of congestion."*

In addition to its simplicity and stateless implementation, another desirable property of CHOKe is that it ensures bounded bandwidth and buffer shares [6] [7]. Specifically, unresponsive flows in CHOKe cannot exceed a certain bandwidth share or buffer limit in the presence of many responsive flows. The following theorems [6] provide an overview of this property, where UDP represents the extreme of unresponsive traffic:

Theorem 1. 1) The maximum UDP bandwidth share in CHOKe is bounded by $(e+1)^{-1} = 26.9\%$.
2) This is attained when UDP input rate, after congestion based dropping, is $C(2e-1)/(e+1) = 1.193C$, where C is the link capacity, and
3) In that case, CHOKe-based UDP dropping rate is $(e-1)/(2e-1) = 38.7\%$.

Theorem 2. When UDP input rate increases without bound, its buffer share approaches 50% but its link utilization approaches 0.

*"Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT. http://www.q2s.ntnu.no

Given these nice properties of CHOKe, it is natural to ask if CHOKe can be generalized and/or if the performance can be improved. Trying to give answers to these questions forms the motivation of this work. Particularly, we aim to generalize CHOKe, while retaining its desirable qualities, i.e., simplicity and statelessness, and simultaneously empower it with tighter control on or some tuning knob to control the use of link bandwidth and buffer space.

In this paper, we introduce geometric CHOKe (gCHOKe), which turns out to be a highly intuitive scheme based on the design principle of CHOKe. A analytical model that characterizes the throughput and buffer occupancy an unresponsive flow can maximally receive under gCHOKe is also presented. The analysis, which is validated through extensive simulation, shows that gCHOKe can achieve over 20% improvement in the bounds of both bandwidth and buffer space used by the aggressive flow. In addition, up to 14% of the total link capacity can be saved from the unresponsive flow, allowing responsive or rate-adaptive flows to obtain a better share of resources in the router. These results provide additional insights into CHOKe.

The rest of this paper is structured as follows. Section 2 outlines the main idea behind gCHOKe. gCHOKe model and assumptions for the analysis are presented in Section 3. Section 4 presents theoretical analysis of UDP throughput and packet loss rates. In Section 5, the model is validated and additional results are given. Section 6 concludes the paper.

2. Geometric CHOKe (gCHOKe)

A. The Scheme

When a packet arrives to a congested queue, gCHOKe randomly samples a packet from the queue. If the incoming and the sampled packet belong to the same flow, gCHOKe continues to sample another random packet from the queue. Matching process stops upon a failed matching trial, or when a pre-defined maximum number of trials is reached. All matched packets and the arriving packet are dropped. In the event of no matching at first attempt, the sampled packet is restored to the queue, but the arriving packet may still be dropped with a probability that depends on the level of congestion. In the case that the buffer is managed by RED, this probability is determined by the RED parameter settings. Throughout this paper, we call such congestion-based dropping probability the *ambient drop rate* or the *RED loss rate*.

The alert reader may have noticed that, by setting the pre-defined maximum number of trials to 1, gCHOKe reduces to CHOKe. Recall that, the main idea and design principle behind CHOKe is that a high-bandwidth unresponsive flow will likely have more packets in the buffer, hence a higher probability of matching and consequently dropping. Due to this, CHOKe punishes the unresponsive flow from possibly dominating the use of the buffer and the link.

The above design principle of CHOKe is inherited by gCHOKe. However, gCHOKe additionally rewards each successful matching with a *bonus trial*. The succession of bonus trials provides an extra shield of protection to rate-adaptive flows from unresponsive ones. By choosing / tuning the defined maximum number of trials, a desired protection level may be achieved. This introduces flexibility for traffic control, which is lacking in the original plain CHOKe.

For ease of presentation and due to space limitation, in the rest of the paper, we do not limit the predefined maximum number of matching trials per arrival. In other words, the number is infinity. This is an extreme case. Any number between 1 and infinity gives a performance between that of the plain CHOKe and this extreme gCHOKe. Similar analysis applies, and the results are omitted.

B. Example Scenario

We compare gCHOKe against CHOKe and RED using simulation of the network setup shown in Fig. 2 where there are $N = 32$ TCP flows. The full simulation parameters are given in Section 5-A. The result (Fig. 1) shows that RED has no fairness mechanism in place. UDP share exceeds 90% of link capacity and this starves out the TCP flows. Under CHOKe, UDP is restricted to 26% of link capacity. Using the enhanced flow protection afforded by gCHOKe, UDP throughput is restricted further down to 19%—a saving of a modest 7% of link capacity C, or an overall improvement of 27% over CHOKe. The improvement can be much higher with higher rate UDP flows (see Fig. 7a).

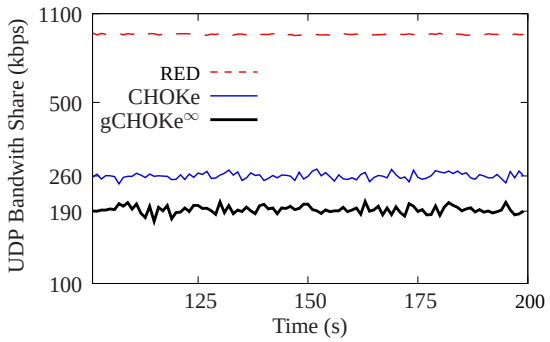


Fig. 1. gCHOKe can restrict high bandwidth flows better than CHOKe and RED.

3. The Model

This section lays out the model and theoretical foundation for the study of steady state performance of gCHOKe.

A. The System and Assumptions

The studied system is shown in Fig. 2. The link capacity is C (pkts/sec) and the buffer is of size b (pkts). We study behavior in the steady state which we assume exists. There is a single unresponsive / aggressive UDP flow and N rate-adaptive similar TCP flows. For analytic simplicity, we consider a gCHOKe queue where the ambient and gCHOKe based droppings are reversed (see also Fig. 4). We highlight that similar model and assumptions have been used for CHOKe analysis [6] [7].

B. Notation

Notations are shown in Table I. Flows are indexed by i, where $i \in (0, N)$. Index 0 denotes the UDP flow, and indices

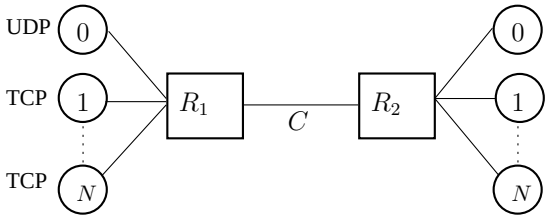


Fig. 2. System model.

$1 \ldots N$ denote TCP flows. To avoid confusion, a hat over a parameter, e.g., $\hat{p}$, is often used to refer to a parameter in the CHOKe scheme.

TABLE I
NOTATION OF STEADY STATE PARAMETERS.

Parameters	Semantics
x_i	source rate of flow i
b_i	flow i packets in buffer
p_i	total flow i drop or loss rate (both RED and gCHOKe)
h_i	the ratio b_i/b (matching probability)
τ	the steady-state queueing delay for TCP flows
b	total backlog $b = \sum_{i=0}^{N} b_i$ in packets
r	congestion or RED-based dropping probability, or ambient drop probability (common to all flows)

C. The Analytical Foundation

The analysis is based on deriving the overall loss / admission probabilities in the steady state for an arbitrary flow.

Consider a flow i whose packet reaches the gCHOKe queue. Since the first dropping is due to RED, by assumption, the packet is dropped with probability r or is admitted by RED with probability $1-r$. After admission by RED, a random packet is sampled from the queue for matching. The flow matching probability is $h_i = b_i/b$. If the packets match, the arriving packet earns a bonus to continue further matching by drawing another packet. This continues until a no-match (or a pre-defined maximum number) is encountered. Assuming that h_i does not change much in the same matching experiment, this results in a sequence of conditional Bernoulli trials. The probability of exactly $k \in [1, \ldots)$ matches is given by $h_i^k(1-h_i)$ and this results in loss of $k+1$ packets from flow i, which are the k matched packets plus the arriving packet. Assuming large b, the total loss probability is,

$$p_i = r + \sum_{k=1}^{\infty} (1-r)(k+1)h_i^k(1-h_i) = r + (1-r)\frac{2h_i - h_i^2}{1-h_i}. \tag{1}$$

A second way to derive the overall drop probability is as follows. Consider an arbitrary packet of flow i. This packet survives both dropping and gets queued to the tail with probability $(1-r)(1-h_i)$. This packet may still be dropped when future packets of the same flow trigger potentially multiple matchings. *How many comparisons or samplings can be tried per each incoming packet? In other words, how many matching trials can be performed before a mismatch happens.* If n comparisons are executed, the first $n-1$ of them must be matching and the last one is the mismatch that aborts the process. This happens with probability $h_i^{n-1}(1-h_i)$. The number of trials is thus a *Geometric* random variable with parameter $q_i = 1-h_i$. The expected number of trials per arriving packet is then $1/q_i = 1/(1-h_i)$. Since a steady state queuing delay τ is assumed, an average of $x_i(1-r)\tau$ flow i packets arrive during τ. A total of $\tau x_i(1-r)/(1-h_i)$ flow matchings can be tried before the enqueued packet gets transmitted. The probability of a trial failing to match the enqueued packet is $(1-1/b)$. Therefore, the overall probability with which a packet survives all droppings is given by the product,

$$1 - p_i = (1-r)(1-h_i)\left(1 - \frac{1}{b}\right)^{\tau x_i(1-r)/(1-h_i)}. \tag{2}$$

4. UDP THROUGHPUT ANALYSIS

Using the gCHOKe model developed in the last section, we are interested in how much bandwidth UDP can "steal" in the presence of many TCP sources. In the analysis, the only independent parameter is the incoming rate x_0 of UDP.

We assume that the gCHOKe link is fully utilized, i.e.,

$$x_0(1-p_0) + Nx_1(1-p_1) = C. \tag{3}$$

For large N,

$$h_1 = \frac{b_1}{b} = \frac{b_1}{b_0 + Nb_1} \leqslant \frac{1}{N} \approx 0. \tag{4}$$

Using (4) in (1), we get for a TCP flow

$$p_1 \approx r. \tag{5}$$

Eqs. (4) and (5) imply that TCP packets seldom trigger matching and are mostly dropped due to ambient dropping.

The approximations (4) and (5) together with (3) are used to derive τ. The number of TCP packets in the buffer is $Nb_1 = b - b_0 = b(1-h_0)$, and the aggregate TCP rate is given by $Nx_1(1-p_1)$. By virtue of Little's Law, we get

$$\tau = \frac{Nb_1}{Nx_1(1-p_1)} = \frac{b(1-h_0)}{C - x_0(1-p_0)}. \tag{6}$$

From (6) and (2), we find for flow 0,

$$1 - p_0 = (1-r)(1-h_0)\left(1 - \frac{1}{b}\right)^{\frac{bx_0(1-r)}{C - x_0(1-p_0)}}$$

For large b, the approximation $(1-1/b)^b \approx e^{-1}$ can be used to simplify the above equation to,

$$1 - p_0 = (1-r)(1-h_0)e^{-\frac{x_0(1-r)}{C - x_0(1-p_0)}} \tag{7}$$

From (1),

$$1 - p_0 = 1 - \left(r + (1-r)\frac{2h_0 - h_0^2}{1-h_0}\right) = (1-r)\frac{h_0^2 - 3h_0 + 1}{1-h_0}. \tag{8}$$

For a packet to be finally transmitted, it must escape both the ambient dropping and successive matching trials from incoming packets. The term $(1-r)$ in (8) corresponds to the probability of surviving the ambient (RED) loss, while the second term $(h_0^2 - 3h_0 + 1)/(1-h_0)$ is the probability

of escaping gCHOKe based packet loss. Therefore, the *pure* gCHOKe based packet loss from a flow with buffer share h_0 is given by (see also (1)):

$$\mathrm{p}_{gCHOKe} = 1 - \frac{h_0^2 - 3h_0 + 1}{1 - h_0} = \frac{2h_0 - h_0^2}{1 - h_0}. \quad (9)$$

For CHOKe, the corresponding drop rate can be shown [6],

$$\mathrm{p}_{choke} = 2\hat{h_0}. \quad (10)$$

Remark: p_{gCHOKe} is a convex function of h_0, and hence, at higher buffer shares, assumes higher drop rates than p_{choke}.

Equating (8) and (7) for flow 0 and simplifying, we obtain

$$\frac{(1-h_0)^2}{h_0^2 - 3h_0 + 1} = \exp\left(\frac{x_0(1-r)}{C - x_0(1-p_0)}\right) = \exp\left(\frac{x_0(1-r)/C}{1 - x_0(1-p_0)/C}\right). \quad (11)$$

Define the UDP utilization μ_0,

$$\mu_0 = x_0(1-p_0)/C. \quad (12)$$

This also means that $x_0/C = \mu_0/(1-p_0)$. Replacing $1-p_0$ by $(1-r)\frac{h_0^2-3h_0+1}{1-h_0}$ from (8), we simplify (11) to:

$$\frac{(1-h_0)^2}{h_0^2 - 3h_0 + 1} = \exp\left(\frac{\frac{\mu_0(1-h_0)}{h_0^2-3h_0+1}}{1-\mu_0}\right) = \exp\left(\frac{\mu_0}{1-\mu_0} \cdot \frac{1-h_0}{h_0^2 - 3h_0 + 1}\right). \quad (13)$$

Define

$$\gamma(h_0) = \frac{1-h_0}{h_0^2 - 3h_0 + 1}. \quad (14)$$

Substituting (14) in (13) and rearranging (13), μ_0 becomes

$$\mu_0 = \frac{\ln[(1-h_0)\gamma(h_0)]}{\gamma(h_0) + \ln[(1-h_0)\gamma(h_0)]} \quad (15)$$

a graphical view of which is shown in Fig. 3.

The next two theorems summarize the above analysis.

Theorem 3. The maximum UDP link utilization is $\mu_0^{max} \approx 20.5\%$, which is achieved when $h_0 \approx 29\%$.

Proof: Eq. (15) has a local maximum within the allowed h_0 range, see also Fig. 3. By setting $\partial\mu_0/\partial h_0 = 0$, we get the maximum numerically at $h_0^* \approx 0.29$, and

$$\mu_0 \leqslant \frac{\ln[(1-h_0^*)\gamma(h_0^*)]}{\gamma(h_0^*) + \ln\left[(1-h_0^*)\gamma(h_0^*)\right]} \approx 0.205.$$

■

Theorem 4. The UDP buffer share in gCHOKe is upper-bounded as $h_0 \leqslant (3-\sqrt{5})/2 \approx 38.2\%$.

Proof: The proof follows from (9) and the fact that $\mathrm{p}_{gCHOKe} = \frac{2h_0 - h_0^2}{(1-h_0)} \leqslant 1$. Solving $2h_0 - h_0^2 \leqslant 1 - h_0$ gives $h_0 \geqslant \frac{3+\sqrt{5}}{2}$ or $h \leqslant \frac{3-\sqrt{5}}{2}$. The first is invalid since h_0 cannot be greater than 1. The proof follows from the second. ■

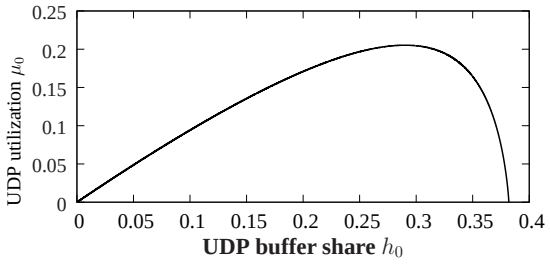


Fig. 3. Eq. (15) in graphical form.

From Ths. 3 and 4, it is easy to see that large UDP buffer occupation does not generally yield high UDP utilization in gCHOKe due to the leaky nature of the queue. This property is a sharp contrast to FIFO. In addition, as h_0 increases from 0.29 to $h_0^{max} = (3-\sqrt{5})/2 \approx 0.382$ where the maximum UDP buffer share is reached, p_{gCHOKe} rapidly increases to 1 and the UDP utilization steadily drops from the maximum possible $\mu_0^* = 0.205$ to near *zero*.

Combining (8), (12) and (14), we find the ratio of UDP traffic admitted by gCHOKe:

$$\frac{\mu_0}{x_0(1-r)/C} = \frac{h_0^2 - 3h_0 + 1}{1 - h_0} = \frac{1}{\gamma(h_0)}. \quad (16)$$

That means, out of $x_0(1-r)$ UDP rate that passes the ambient drop, $\mu_0 C$ are transmitted by gCHOKe and $x_0(1-r) - \mu_0 C$ are dropped by gCHOKe. The rate of gCHOKe based dropping is then,

$$\frac{x_0(1-r) - \mu_0 C}{x_0(1-r)} = 1 - \frac{\mu_0 C}{x_0(1-r)} = 1 - \frac{1}{\gamma(h_0)} = \frac{2h_0 - h_0^2}{1-h_0} = \mathrm{p}_{gCHOKe}. \quad (17)$$

It follows that $1/\gamma(h_0)$ and p_{gCHOKe} are gCHOKe's admission and dropping rates respectively.

A summary of the theoretical results for the gCHOKe model is shown in Table II. The table is useful for understanding the relationships between the various parameters of the model. Many of them are self-explanatory and can be obtained from the formulae above. The "I/O(input/output)" in the last but one row refers to the relation between the input UDP traffic after ambient dropping, i.e., $x_0(1-r)$ and the UDP utilization or transmission rate μ_0, which follows readily from (16).

Before concluding this section, we remark that the reversal of RED and gCHOKe based droppings has small impact on overall drop probability p_0. If we followed the actual gCHOKe depicted in Fig. 4, instead of (1), we would have obtained,

$$p_0 = \mathrm{p}_{gCHOKe} + r(1-h_0) = \frac{2h_0 - h_0^2}{1-h_0} + r(1-h_0) \quad (18)$$

The difference between (18) and (1) is $r(\mathrm{p}_{gCHOKe} - h_0)$ which is insignificant, largely due to small r. Compared to that in RED, r in gCHOKe is generally smaller. The reason is that since gCHOKe drops excessively in response to increasing input x_0, the average queue length `avg` is less in gCHOKe. This in turn keeps r smaller and makes the p_0 approximation in (1) reasonably accurate.

TABLE II
SUMMARY OF THEORETICAL RESULTS IN STEADY STATE.

Parameter	gCHOKe Model Formulation
Intermediate variable	$\gamma(h_0) = \frac{1-h_0}{(h_0^2-3h_0+1)}$
gCHOKe drop date	$p_{gCHOKe} = \frac{2h_0-h_0^2}{1-h_0} = 1-1/\gamma(h_0)$
UDP utilization	$\mu_0 = \frac{\ln[(1-h_0)\gamma(h_0)]}{\gamma(h_0)+\ln[(1-h_0)\gamma(h_0)]}$
UDP I/O relation	$x_0(1-r)/C = \mu_0\gamma(h_0)$
UDP admission rates $(1-p_0)$	$(1-r)(1-p_{gCHOKe}) = (1-r)/\gamma(h_0)$ $= (1-r)(1-h_0)(1-\frac{1}{b})^{\tau x_0(1-r)/(1-h_0)}$

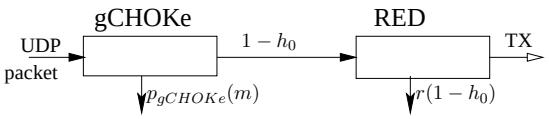


Fig. 4. Actual gCHOKe scheme.

5. Model Validation and Simulation Results

A. Simulation Environment and Default Settings

We have implemented gCHOKe and all simulations are performed in ns2-34. Unless otherwise stated, the following simulation parameters[1] are used: The bottleneck link has a capacity of $C = 1Mbps$, a latency of 1ms and a buffer capacity $b = 300kB$. RED buffer thresholds are set as $\min_{th} = 20$ and $\max_{th} = 300$. We use $N = 32$ TCP-SACK flows and all packets are 1000 bytes. Flow start times are random and uniformly distributed on [0,10]. All simulations are replicated 20 times and each runs for 200s, but only the results of the last 100s are taken. The 95% confidence intervals are very small and hence are not reported.

B. Model Validation

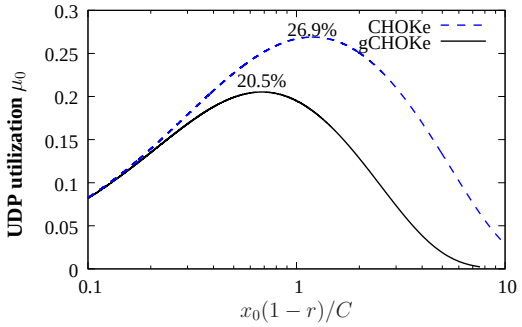
As explained before, the only independent parameter of the model is the input UDP traffic rate x_0. In Fig. 5, we plot theoretical UDP utilizations and buffer shares as we vary the input UDP rate $x_0(1-r)$ from $0.1C$ to $10C$. It is trivial to generate these (indeed all) theoretical plots from parameter interdependencies summarized in Table II. As can be seen from simulation results in Fig. 6, the model is very accurate.

C. Results and Discussion

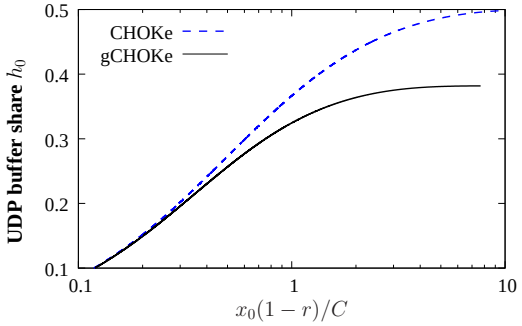
From the theoretical and matching simulation results, we highlight the following salient points:

- UDP utilization bounds: CHOKe (26.9%) and gCHOKe (20.5%); UDP buffer share bounds: CHOKe (50%) and gCHOKe (38.2%). Both gCHOKe bounds are tighter by over 20% over CHOKe's.
- For low and moderate UDP traffic rate, there is no significant difference between CHOKe and gCHOKe in terms of UDP utilization or buffer share levels.

[1]A parallel set of experiments with $C = 20Mbps$, $b = 1MB$, $\min_{th} = 20$, $\max_{th} = 1000$, $N = 100$ gives statistically consistent results.



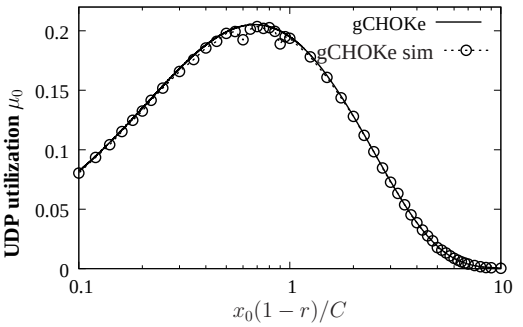
(a) u_0 versus $x_0(1-r)/C$.



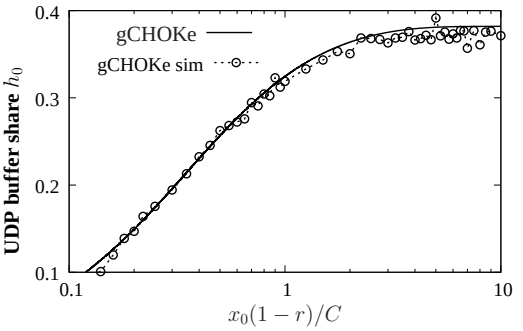
(b) h_0 versus $x_0(1-r)/C$.

Fig. 5. UDP share u_0 and h_0 *vs.* UDP incoming rate x_0.

- Increasing UDP input rate initially increases the buffer share, but does not allow buffer monopoly (see Fig. 5b). UDP's buffer shares eventually stabilize around their peaks (see also Fig. 3).



(a) gCHOKe μ_0



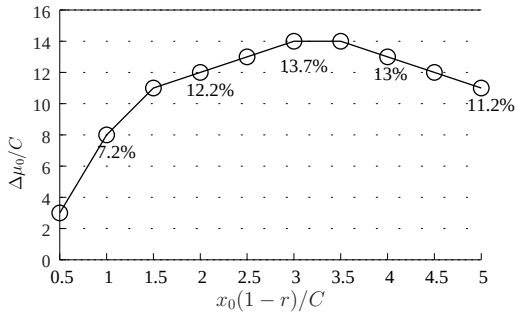
(b) gCHOKe h_0

Fig. 6. Model Validation: μ_0 and h_0 under gCHOKe.

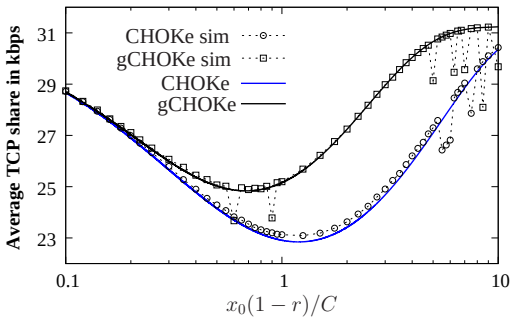
- In sharp contrast to Drop Tail and RED (e.g., see Fig. 1), while indefinitely increasing UDP input traffic will increase the corresponding buffer share, it may not yield higher utilization for UDP in both CHOKe an gCHOKe (see Fig. 5a). Particularly, increasing the input rate beyond $0.682C$ backfires in gCHOKe since each incoming UDP packet potentially triggers a sequence of successful comparisons at soaring rate. UDP utilization peaks at 20.5% in gCHOKe (*vs.* 26.9% in CHOKe). This occurs when 29% of the packets in buffer are UDP (*vs.* 39% for CHOKe). This means, as UDP buffer ratio increases from 29% to 38.2% due to increased input, $p_{gCHOKe} \to 1$ and UDP utilization drops off from maximum 20.5% to almost 0.
- The high UDP buffer occupancy but low UDP utilization (see Fig. 5a and Fig. 5b) at high incoming UDP rate implies nonuniform UDP packet distribution in the queue where most of the UDP packets may have been clustered closer to the tail of the queue.

In view of the last property, it is easy to see that packets enter the queue tail at a rate $[x_0(1-h_0)+Nx_1(1-h_1)](1-r)$, but exit only at the rate of C. When x_0 is high, since $h_1 \approx 0$ (see Eq. 4) and r is realistically insignificant, most of the UDP packets get dropped as they advance towards the head of the queue. Assuming a fluid model, we say the UDP flow is *thinned* by flow matching. The power of thinning is intrinsic to the scheme and is unique to a flow rate x_0. The flow controlling or thinning power of the schemes is conveniently captured by Eq. (2). An arriving UDP packet gets enqueued at queue tail with probability $(1-r)(1-h_0)$. The UDP flow gets thinned due to flow matching by factors of $(1-1/b)^{\tau x_0(1-r)}$ and $(1-1/b)^{\tau x_0(1-r)/(1-h_0)}$, respectively for CHOKe and gCHOKe, before the flow packet can reach the head of the queue. When h_0 is excessively high, due to high input UDP traffic rate x_0, the thinning exponent of gCHOKe increases faster. In the extreme case when $h_0^{max} = 0.382$, each arriving UDP packet in gCHOKe performs, on average, $1/(1-h_0^{max}) \approx 1.62$ matching trials. This allows gCHOKe to restrict the UDP bandwidth and buffer shares to lower levels.

Fig. 7a presents the difference between the UDP utilization levels that can go under CHOKe and gCHOKe. Note that while it is difficult to derive $\Delta\mu = \mu_{choke}(x_0) - \mu_{gCHOKe}(x_0)$ for a given x_0 in closed form, UDP utilization values can be computed numerically (also from Fig. 5a) for a given input rate x_0. The resulting $\Delta\mu_0$ is plotted in percentage of C for selected input rates in Fig. 7a. As can be seen, gCHOKe wields better control of high bandwidth flows. Up to 14% of the total link bandwidth can be saved by using gCHOKe! As a consequence, gCHOKe provides better throughput to TCP flows, as shown by Fig. 7b. The simulation results generally fit the theoretical TCP throughput (see Eq. 3) except for some dips especially at higher input UDP rates. We believe that this is due to approximation errors in TCP dropping probabilities, see Eq. 5. As TCP gains more throughput, the incidence of successful TCP flow matching increases. This in turn results in occasional packet losses and drops in TCP bandwidths.



(a)



(b)

Fig. 7. More bandwidth is available for TCP Flows under gCHOKe.

6. Conclusion

Regardless of the outcome of the experiment, CHOKe maximally triggers a single trial of flow matching per packet arrival. Therefore, flow protection in CHOKe is flat. In this paper, we present gCHOKe, a generalization of CHOKe, which provides additional power of protection and at the same time retains the statelessness and simplicity of CHOKe. gCHOKe rewards a successful flow matching with a bonus trial. Depending on the recent admission history of the flow, or the ratio of flow packets queued in the buffer, a single packet arrival may trigger a succession of packet droppings. With this way, a flow with relatively many recent arrivals can be controlled further, leaving the network resources (buffer and link bandwidth) to responsive flows. Compared to CHOKe, the flow protection performance of gCHOKe is superior ensuring tighter upper bounds in the use of network resources by a UDP flow.

References

[1] P. Goyal, H. M. Vin, and H. Cheng, "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE/ACM Trans. on Networking (ToN)*, vol. 5, no. 5, 1997.

[2] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM ToN*, vol. 1, no. 4, 1993.

[3] D. Lin and R. Morris, "Dynamics of random early detection," in *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4, 1997.

[4] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *IEEE ICNP*, 2001.

[5] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe - a stateless active queue management scheme for approximating fair bandwidth allocation." in *IEEE Infocom*, 2000.

[6] J. Wang, A. Tang, and S. H. Low, "Maximum and asymptotic udp throughput under CHOKe," in *ACM SIGMETRICS*, 2003.

[7] A. Tang, J. Wang, and S. H. Low, "Understanding CHOKe: Throughput and Spatial Characteristics," *IEEE/ACM ToN*, vol. 12, no. 4, 2004.